Binary Classification Model

Table of Contents

Objectives

The main objective of this report is to use the yellow taxicab dataset to build a binary classification model that can predict whether a trip will have a toll or not. Since toll cost adds to the total amount billed, this model will help the business to estimate the accurate total amount that a customer will be charged on any trip.

We will use different classification metrics to select the best model.

Brief description of the data set and a summary of its attributes

TLC Trip Record Data has 12 years (2009 –2020) worth of Data available but I have decided to analyze a subset of the 2015.

In 2015, passengers took nearly 300 million yellow cab rides in New York City. Working with the complete dataset for all these rides would require considerable time and computational resources. The 12 data files used represent two percent of the total trips sampled at random from each month.

I chose this data because it is useful for real-world applications such as:

- Predicting taxi duration for a trip
- Allocating taxi to zones/regions based on demand.
- Identify whether a toll fee will be paid

Below is the summary of the data attributes as described here

Attribute / Column	Description
VendorID	A code indicating the TPEP provider that provided the record. **1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.**
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
RateCodeID	The final rate code in effect at the end of the trip. **1= Standard rate** **2=JFK** **3=Newark** **4=Nassau or Westchester** **5=Negotiated fare**

	6=Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. **Y= store and forward trip** **N= not a store and forward trip**
Payment_type	A numeric code signifying how the passenger paid for the trip. **1= Credit card** **2= Cash** **3= No charge** **4= Dispute** **5= Unknown** **6= Voided trip**
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount –This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

Data Exploration

Joining the 12 files provided resulted in 2, 922, 266 instances in the dataset with no missing values. We have 12 numerical features, 4 categorical features, 2 datetime features and 1 integer. For easy visualization, I converted *VendorID*, *RateCodeID and payment_type* to their corresponding values.

Summary of some the numerical attributes shows that this data is not perfect and therefore needs some cleaning for example there is no way we can have a negative tip or zero passengers.

	Passenger_count	Trip_distance	Fare_amount	Tip_amount
min	0.0	0.0	-150.00	-2.7
max	9.0	14680110.0	410266.86	650

Data Cleaning and Feature engineering

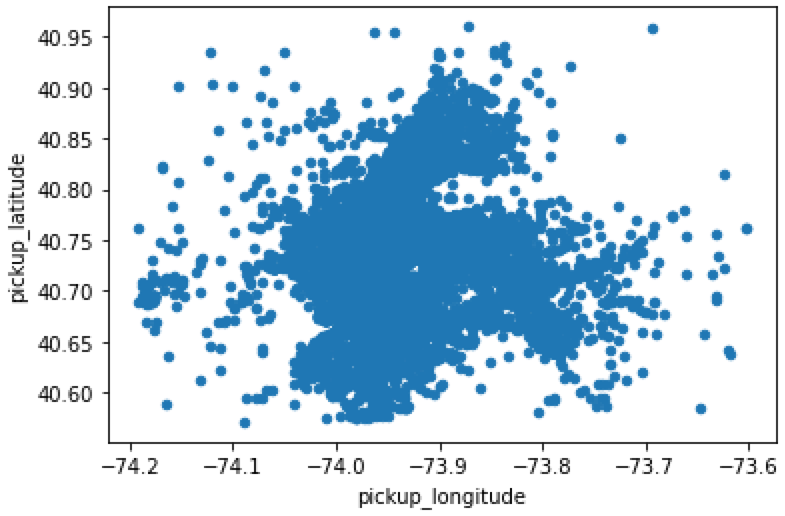
For data preprocessing/cleaning

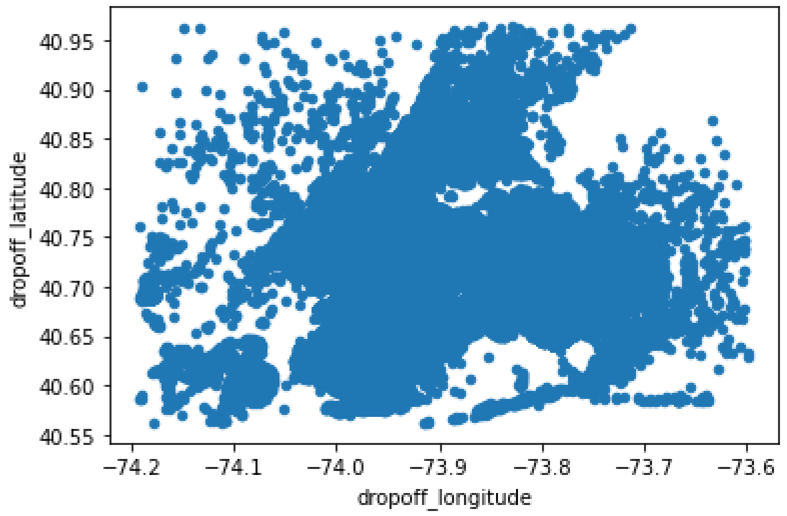
- Charges and trip information that are negative, as well as charges inconsistent with expected values are considered incorrect. In addition, trips with pickup or drop off locations outside a geographic region of interest are removed.
- Only keep trips with valid passenger and distance information.
- Remove trips missing valid pickup or drop off locations.
- Remove trips with invalid Rate code.

For Feature engineering, the following features were added.

- ***duration*** - Length of the trip, in minutes, calculated from the pickup and drop off times.
- ***avespeed*** - Average speed, in mph, calculated from the distance and duration values.
- ***time_of_day* -** This feature represents pick up time as the elapsed time since midnight in decimal hours (e.g. 7:10 am becomes 7.1667). The output is a duration vector with units of hours.
- ***day_of_week*** - This feature is a categorical array indicating the day of the week the trip began, in long format (e.g. 'Monday').
- ***toll_paid* –** This feature indicates trips that charged a toll or not.

After removing invalid trip information, we can now visualize some of the features.



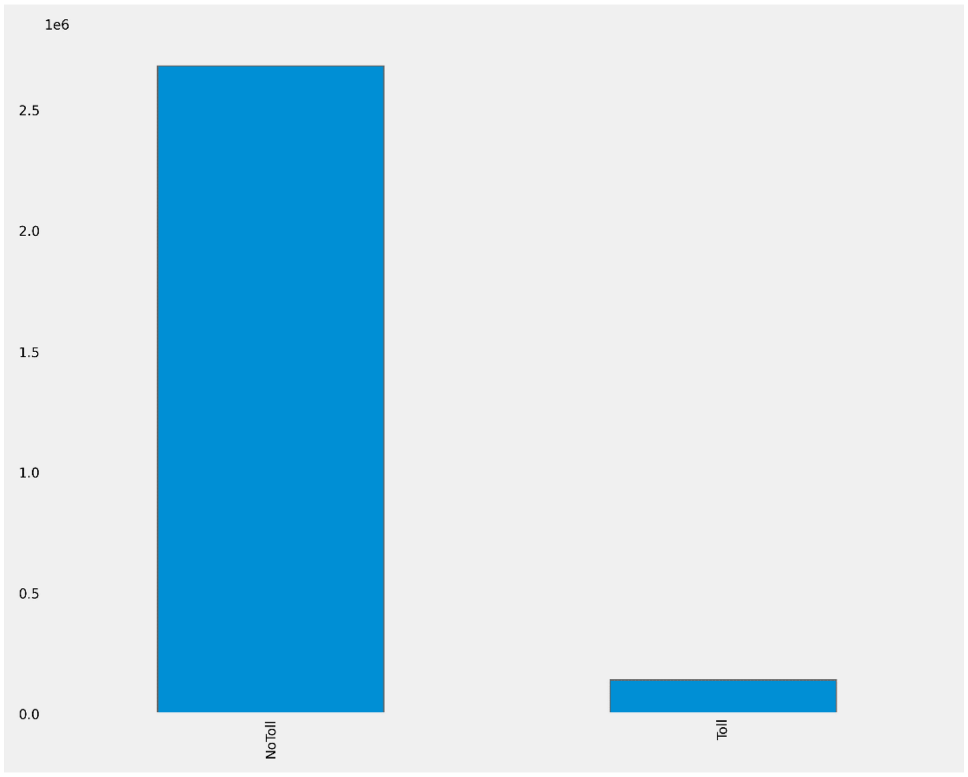


Data Preparation for Machine Learning

Target Variable

We chose the target variable to be ***toll_paid*** as we want to predict the presence or absence of toll(s) for a trip.

	Count	**Percentage (%)**
No Toll	2682515	0.95
Toll	141186	0.05
	2823701	**100**



The distribution above shows that we have unbalanced classes as 95% of the data belongs to *No Toll* class.

Feature Selection

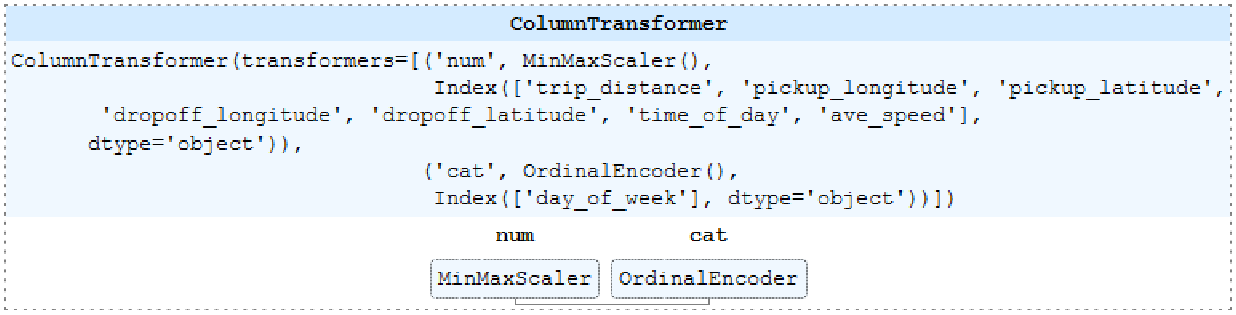
To avoid using all the 22 columns as features, we tried to find the correlation between each numerical feature and the tolls_amount, and the result is as below:

Features	Correlation with tolls_amount
total_amount	0.675407
trip_distance	**0.634744**
fare_amount	0.606455
duration	0.457660
pickup_longitude	**0.434626**
tip_amount	0.431183
ave_speed	**0.412052**
dropoff_longitude	**0.303816**
passenger_count	0.012695
time_of_day	**0.001991**
dropoff_latitude	**-0.054995**
extra	-0.073699
pickup_latitude	**-0.111228**

We opted to use the highlighted numerical features and ***day_of_week*** as the only categorical feature, a total of 8 features.

Feature transformation

We applied two transformations on the features: feature scaling using *MinMax scaler* for the numerical features and ordinal encoding for the categorical feature. We built below transformation pipeline using scikit-learn's ***ColumnTransformer*** class.

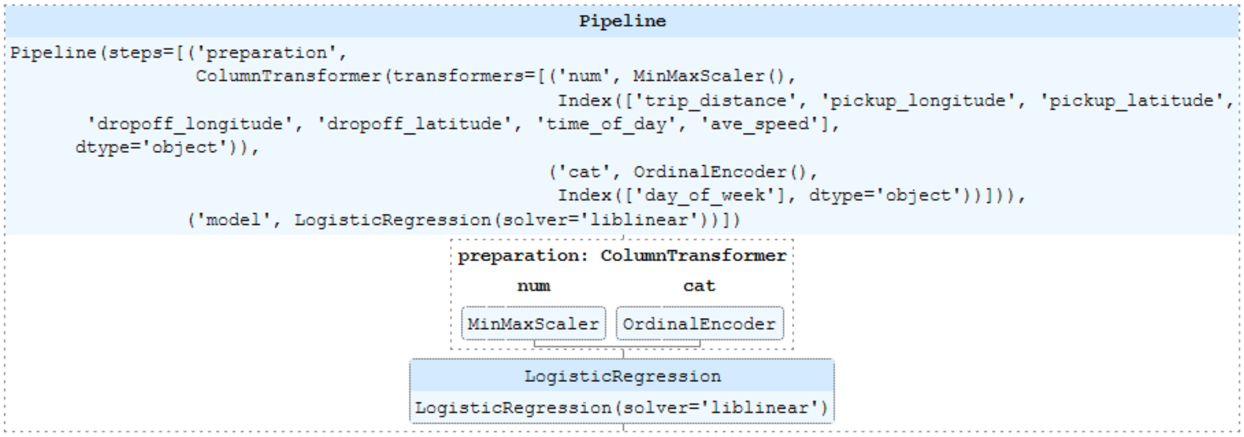


```
ColumnTransformer
ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                 Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                ('cat', OrdinalEncoder(),
                                 Index(['day_of_week'], dtype='object'))])
```

Model Training and Evaluation

We trained ten different classification models:

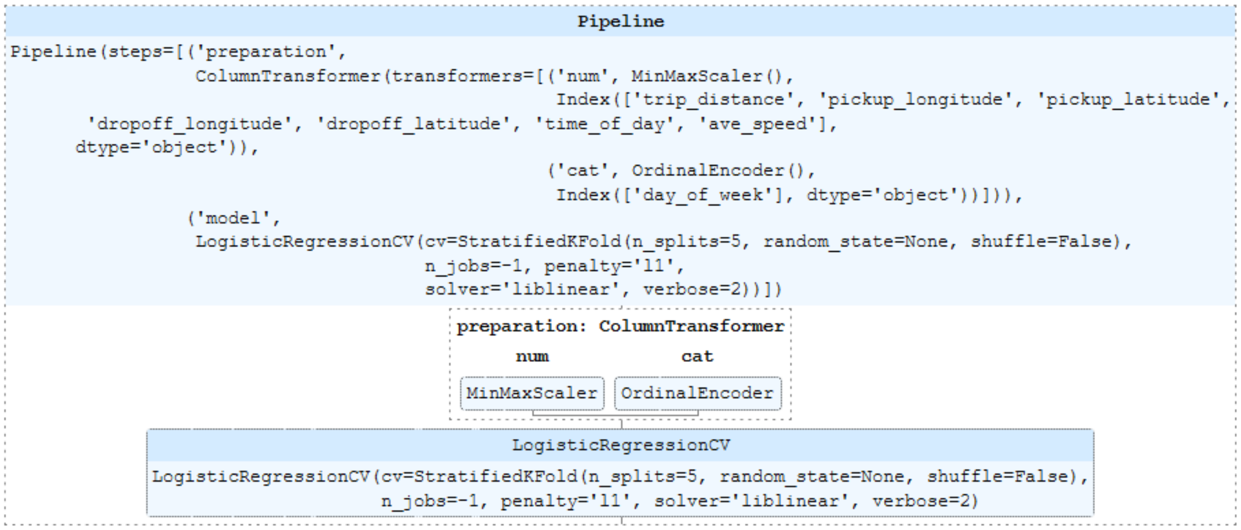
- Simple Logistic Regression with default parameters



```
Pipeline
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model', LogisticRegression(solver='liblinear'))])
```

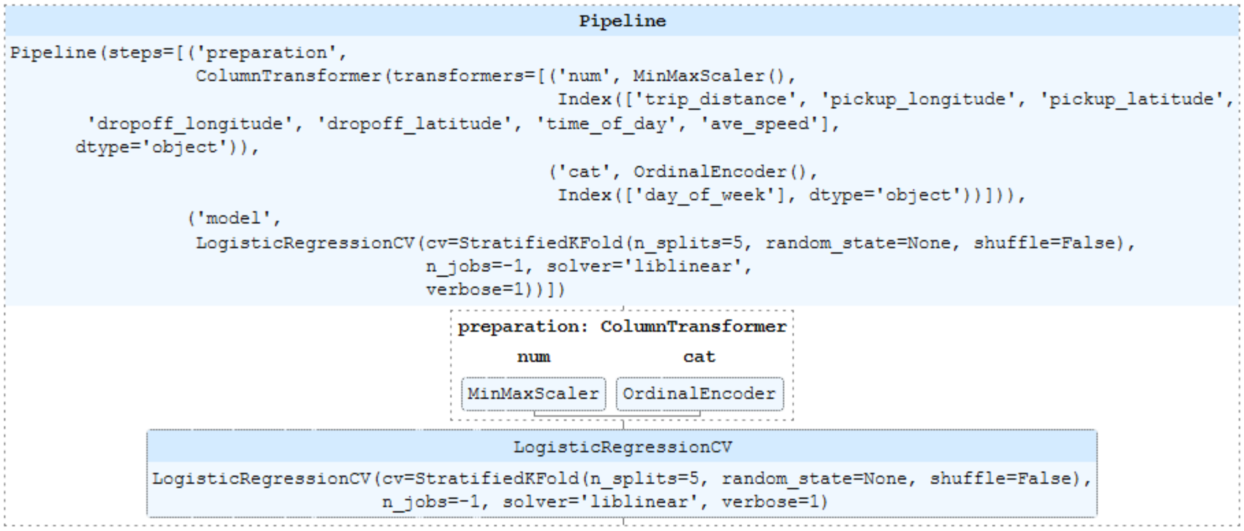
preparation: ColumnTransformer
num
cat
MinMaxScaler
OrdinalEncoder
LogisticRegression
LogisticRegression(solver='liblinear')

- Logistic Regression CV with stratified KFold cross validation and l1 penalty



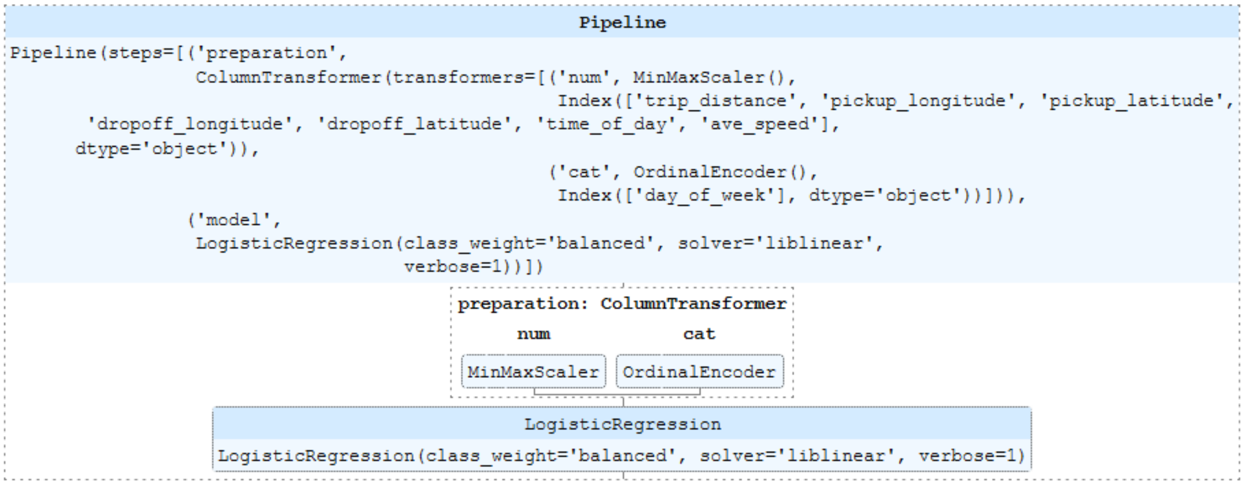
```
Pipeline
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model',
                 LogisticRegressionCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
                                      n_jobs=-1, penalty='l1',
                                      solver='liblinear', verbose=2))])
preparation: ColumnTransformer
num                cat
MinMaxScaler   OrdinalEncoder
LogisticRegressionCV
LogisticRegressionCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
                     n_jobs=-1, penalty='l1', solver='liblinear', verbose=2)
```

- Logistic Regression CV with stratified KFold cross validation and l2 penalty



```
Pipeline
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model',
                 LogisticRegressionCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
                                      n_jobs=-1, solver='liblinear',
                                      verbose=1))])
preparation: ColumnTransformer
num                cat
MinMaxScaler   OrdinalEncoder
LogisticRegressionCV
LogisticRegressionCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
                     n_jobs=-1, solver='liblinear', verbose=1)
```

- Logistic Regression with class weight parameter as balanced



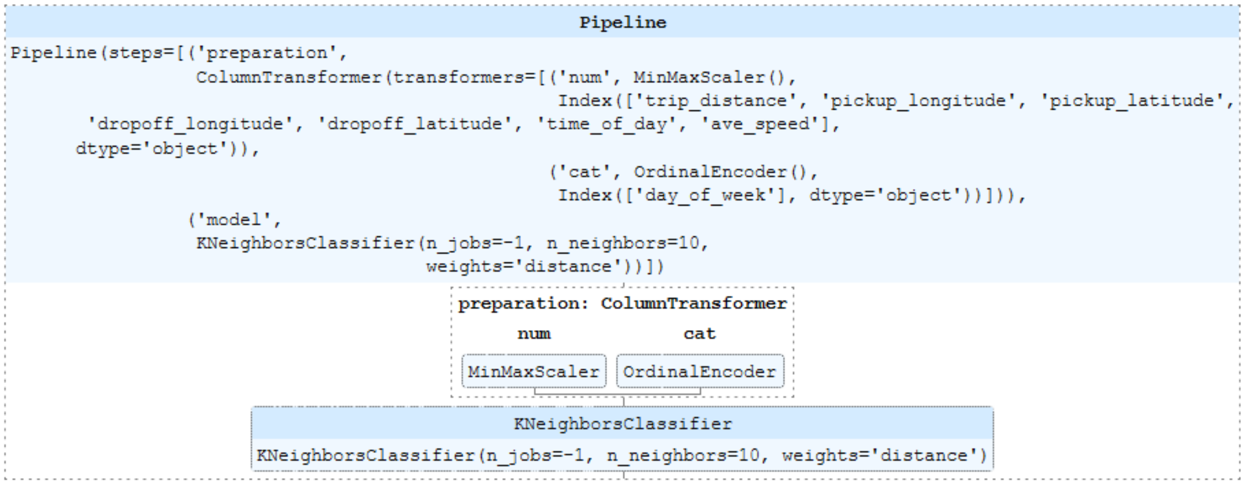
```
Pipeline

Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model',
                 LogisticRegression(class_weight='balanced', solver='liblinear',
                                    verbose=1))])

preparation: ColumnTransformer
num                cat
MinMaxScaler       OrdinalEncoder

LogisticRegression
LogisticRegression(class_weight='balanced', solver='liblinear', verbose=1)
```

- K-Nearest Neighbors with k=10



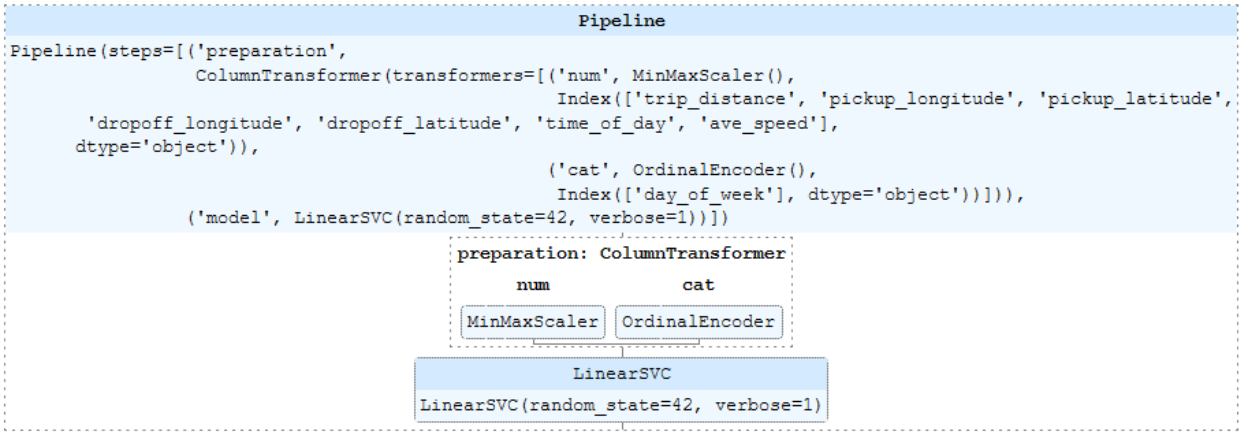
```
Pipeline

Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model',
                 KNeighborsClassifier(n_jobs=-1, n_neighbors=10,
                                      weights='distance'))])

preparation: ColumnTransformer
num                cat
MinMaxScaler       OrdinalEncoder

KNeighborsClassifier
KNeighborsClassifier(n_jobs=-1, n_neighbors=10, weights='distance')
```

- Support Vector Machine (Linear SVC)

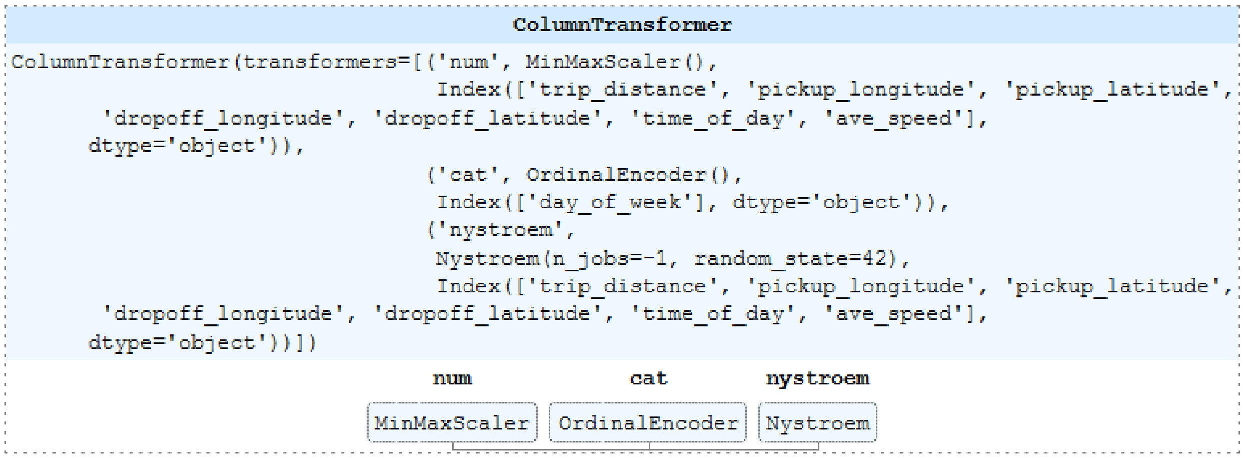


Pipeline

```
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model', LinearSVC(random_state=42, verbose=1))])
```

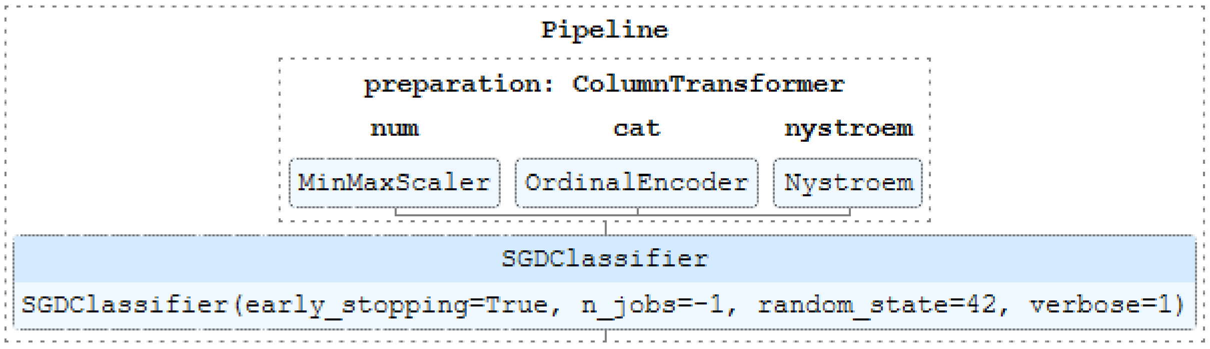
```
LinearSVC(random_state=42, verbose=1)
```

- SGD Classifier with nystroem preprocessing



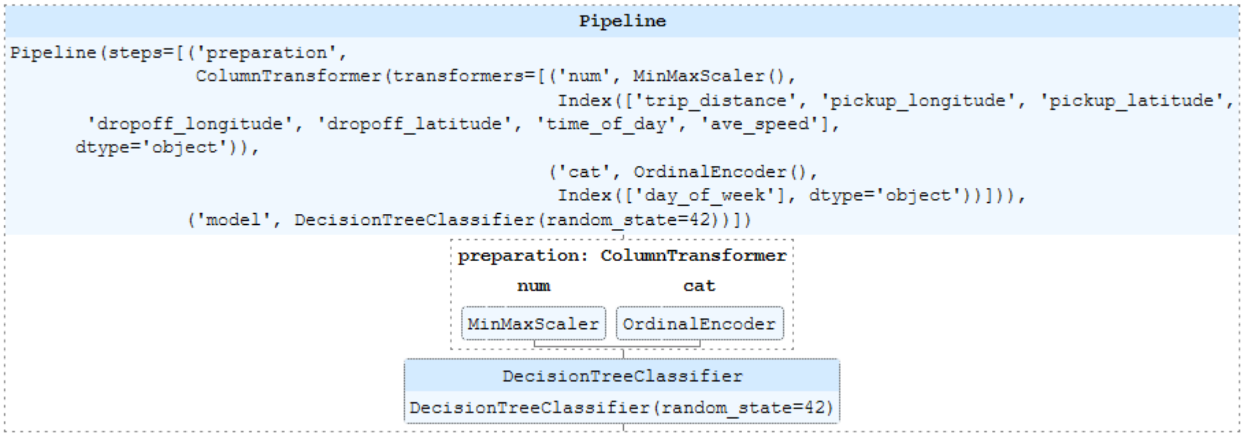
ColumnTransformer

```
ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                 Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                ('cat', OrdinalEncoder(),
                                 Index(['day_of_week'], dtype='object')),
                                ('nystroem',
                                 Nystroem(n_jobs=-1, random_state=42),
                                 Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object'))])
```



```
SGDClassifier(early_stopping=True, n_jobs=-1, random_state=42, verbose=1)
```

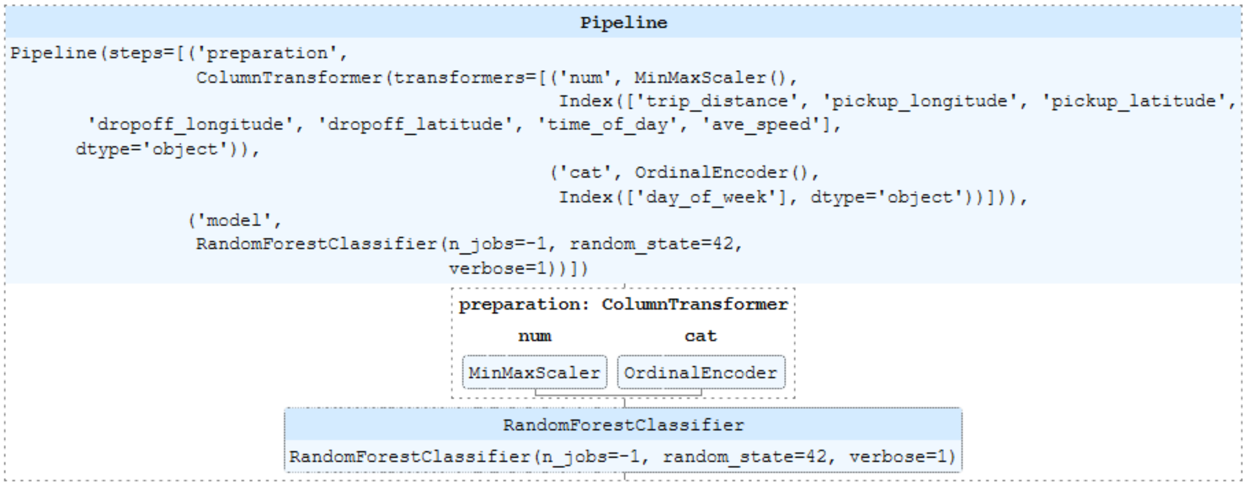
- Decision Tree Classifier



```
Pipeline
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model', DecisionTreeClassifier(random_state=42))])
```

preparation: ColumnTransformer
num
cat
MinMaxScaler
OrdinalEncoder
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)

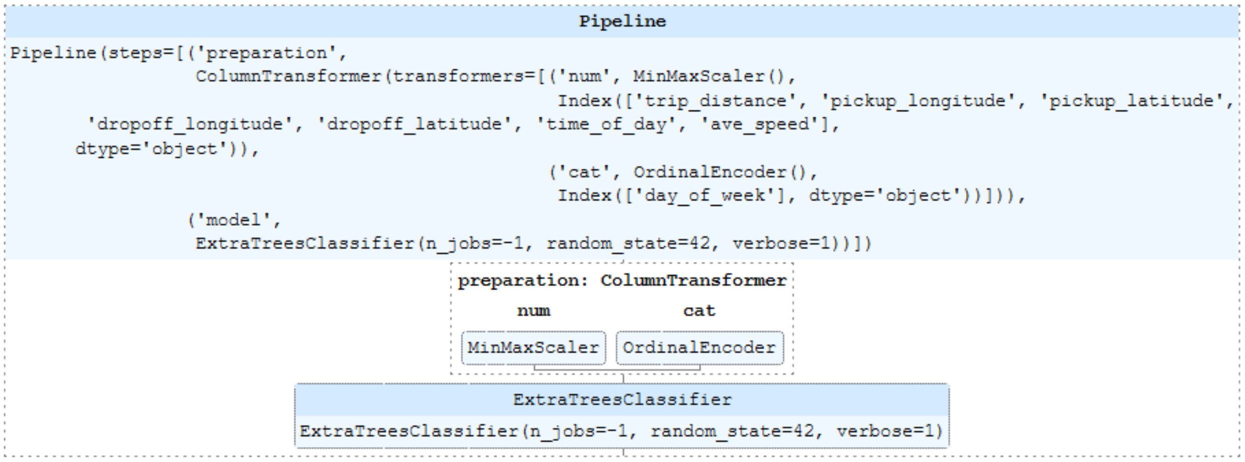
- Random Forest Classifier



```
Pipeline
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model',
                 RandomForestClassifier(n_jobs=-1, random_state=42,
                                        verbose=1))])
```

preparation: ColumnTransformer
num
cat
MinMaxScaler
OrdinalEncoder
RandomForestClassifier
RandomForestClassifier(n_jobs=-1, random_state=42, verbose=1)

- Extra Tree Classifier



```
Pipeline
Pipeline(steps=[('preparation',
                 ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                                  Index(['trip_distance', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'time_of_day', 'ave_speed'],
      dtype='object')),
                                                 ('cat', OrdinalEncoder(),
                                                  Index(['day_of_week'], dtype='object'))])),
                ('model',
                 ExtraTreesClassifier(n_jobs=-1, random_state=42, verbose=1))])
```

preparation: ColumnTransformer
num
cat
MinMaxScaler
OrdinalEncoder
ExtraTreesClassifier
ExtraTreesClassifier(n_jobs=-1, random_state=42, verbose=1)

We split our data into two (train: 2,258,960 rows and test: 564, 741 rows) using sklearn's StratifiedShuffleSplit class to ensure that the distribution of each class is represented accurately.

In addition to accuracy, we evaluated the models using below metrics given the fact that we are dealing with unbalanced classes.

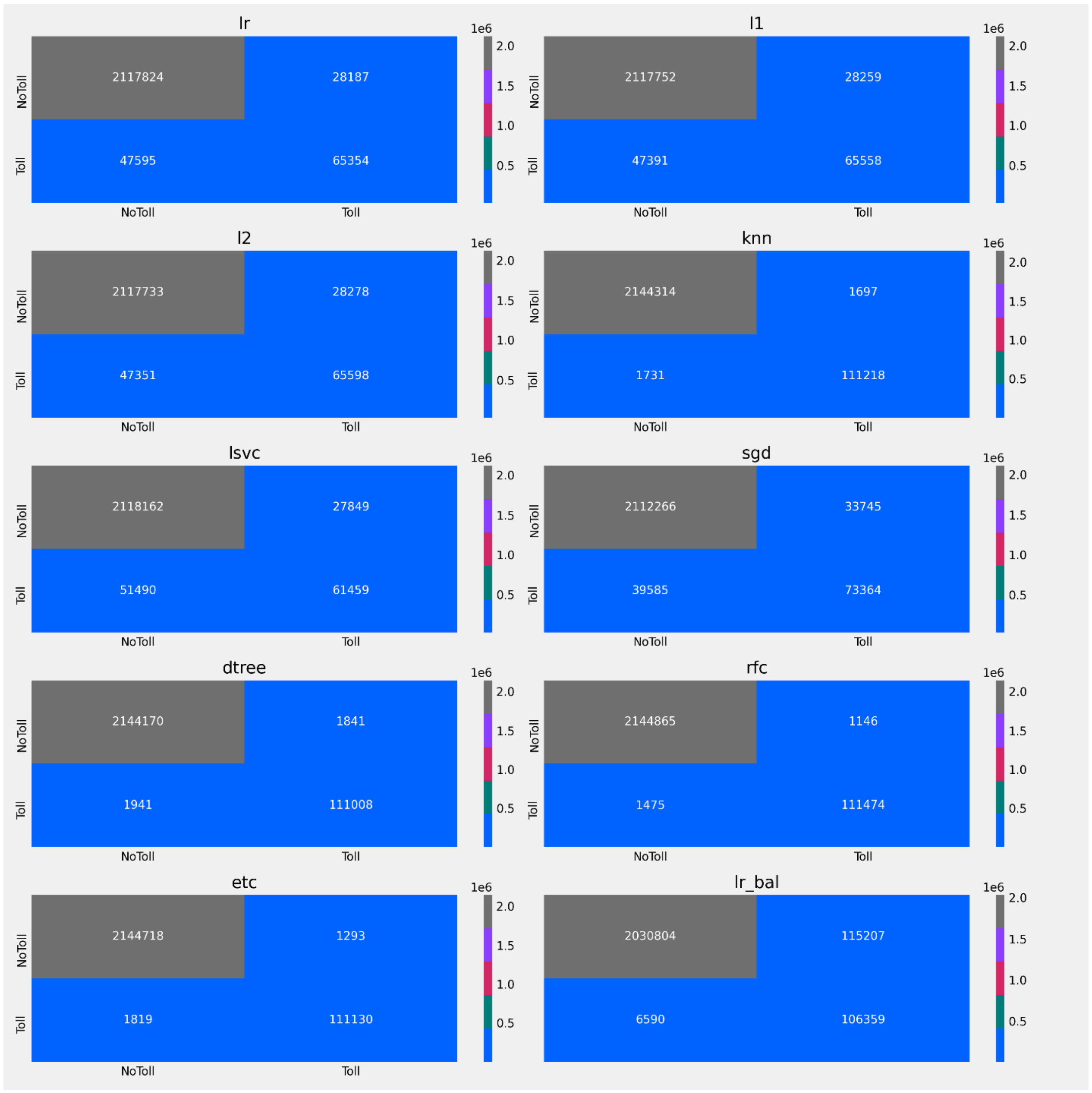
	Logistic Regression		**Logistic Regression CV (penalty=l1)**		**Logistic Regression CV (penalty=l2)**		**Logistic Regression (weight=balanced)**	
	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	0.966453	0.966409	0.966511	0.966491	0.966520	0.966512	0.946083	0.946367
Precision	0.698667	0.699849	0.698786	0.700254	0.698773	0.700357	0.480033	0.481373
Recall	0.578615	0.574636	0.580421	0.576655	0.580775	0.577186	0.941655	0.939016
F1 Score	0.632999	0.631092	0.634127	0.632472	0.634333	0.632834	0.635900	0.636469
Auc	0.782740	0.780833	0.783627	0.781832	0.783799	0.782094	0.943985	0.942885

	K-Nearest Neighbors(k=10)		**LinearSVC**		**SGD Classifier**	
	Train	Test	Train	Test		
Accuracy	**0.998482**	**0.987759**	0.964878	0.964748	0.967538	0.967571
Precision	**0.984971**	**0.879647**	0.688169	0.688738	0.684947	0.686516
Recall	**0.984674**	**0.874880**	0.544131	0.538195	0.649532	0.646740
F1 Score	**0.984823**	**0.877257**	0.607732	0.604230	0.666770	0.666035
Auc	**0.991942**	**0.934290**	0.765577	0.762697	0.816904	0.815598

	Decision Tree		**Random Forest**		**Extra Tree Classifier**	
	Train	Test	Train	Test	Train	Test
Accuracy	**0.998326**	**0.986236**	**0.998840**	**0.990656**	**0.998622**	**0.988972**
Precision	**0.983686**	**0.864933**	**0.989824**	**0.914620**	**0.988499**	**0.904443**
Recall	**0.982815**	**0.858838**	**0.986941**	**0.896837**	**0.983895**	**0.871516**
F1 Score	**0.983251**	**0.861875**	**0.988380**	**0.905641**	**0.986192**	**0.887674**
Auc	**0.990979**	**0.925890**	**0.993203**	**0.946216**	**0.991646**	**0.933335**

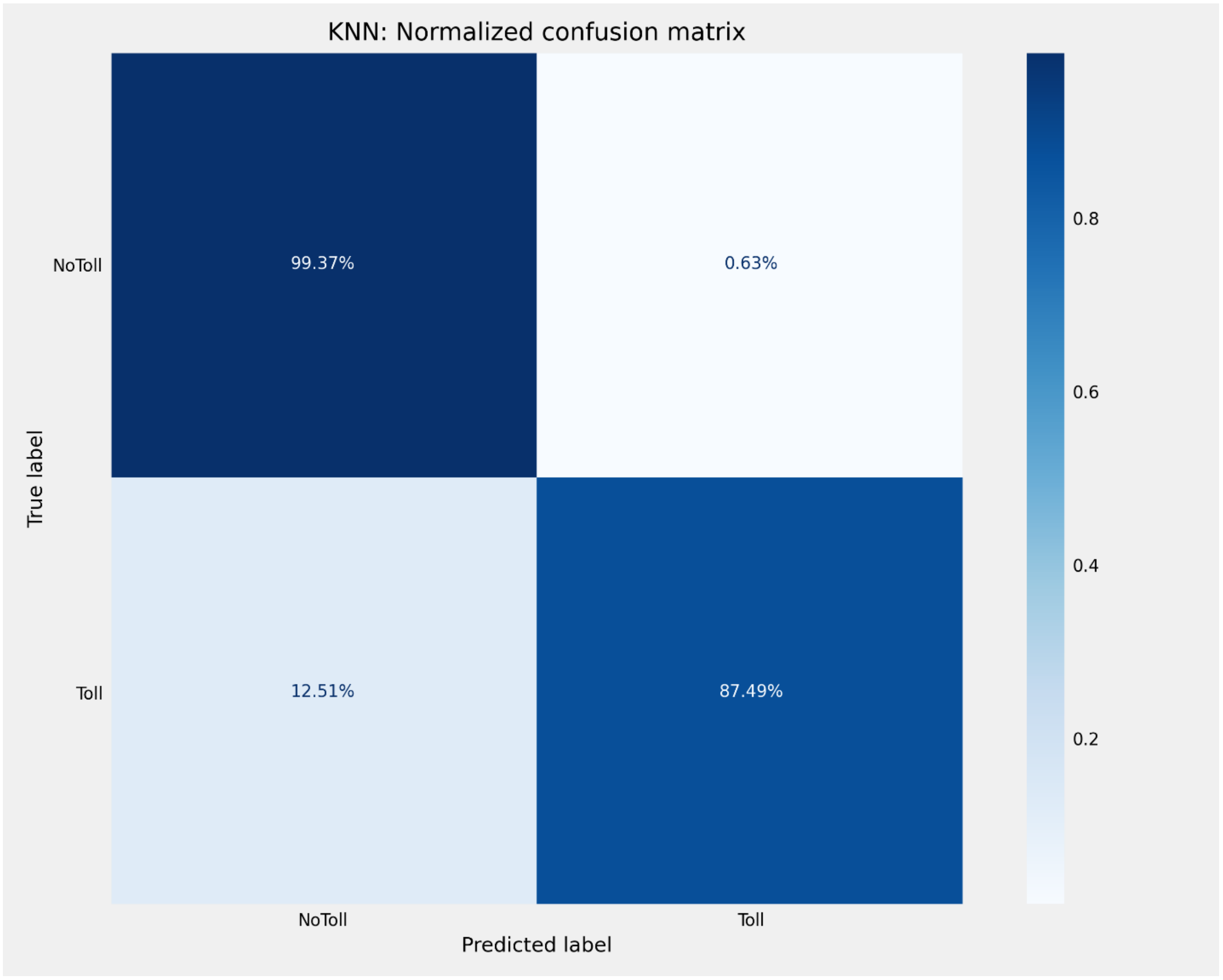
Key Findings and Insights

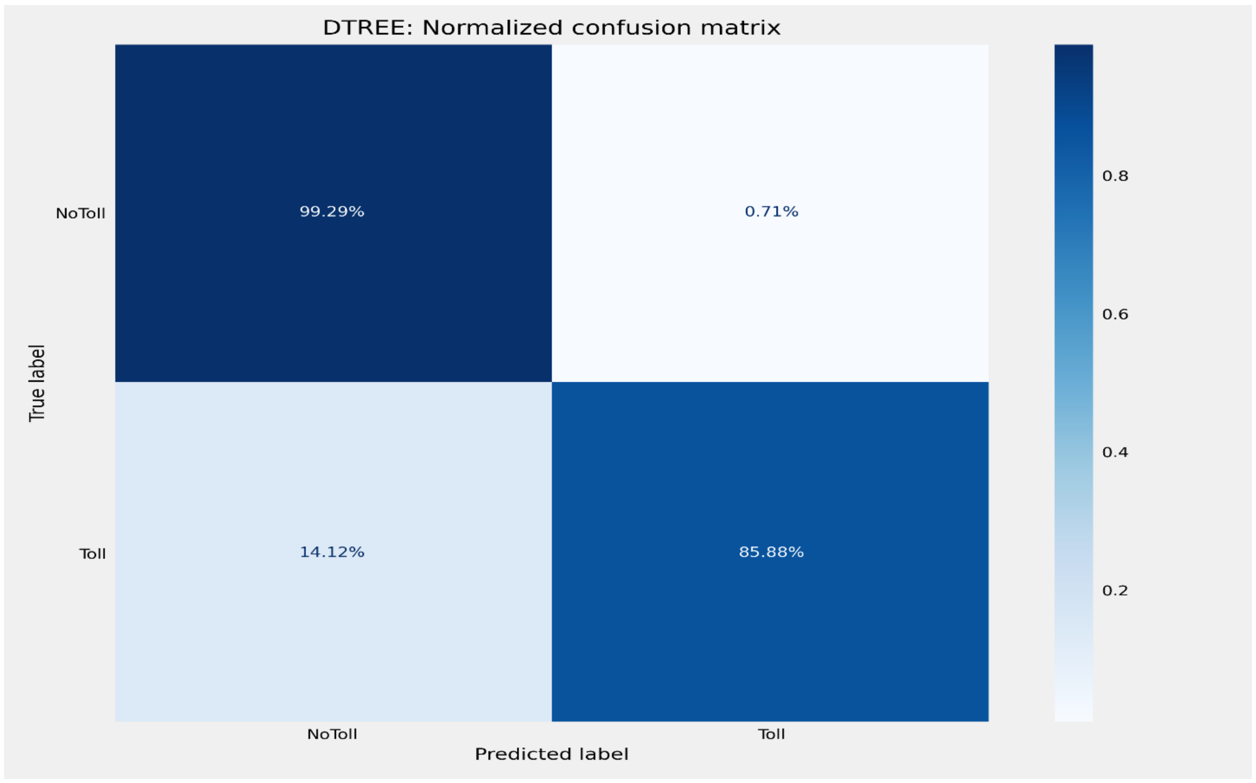
We explored the confusion matrix of all the models to understand the number and percentage of the two classes correctly predicted given the true values for the training data.

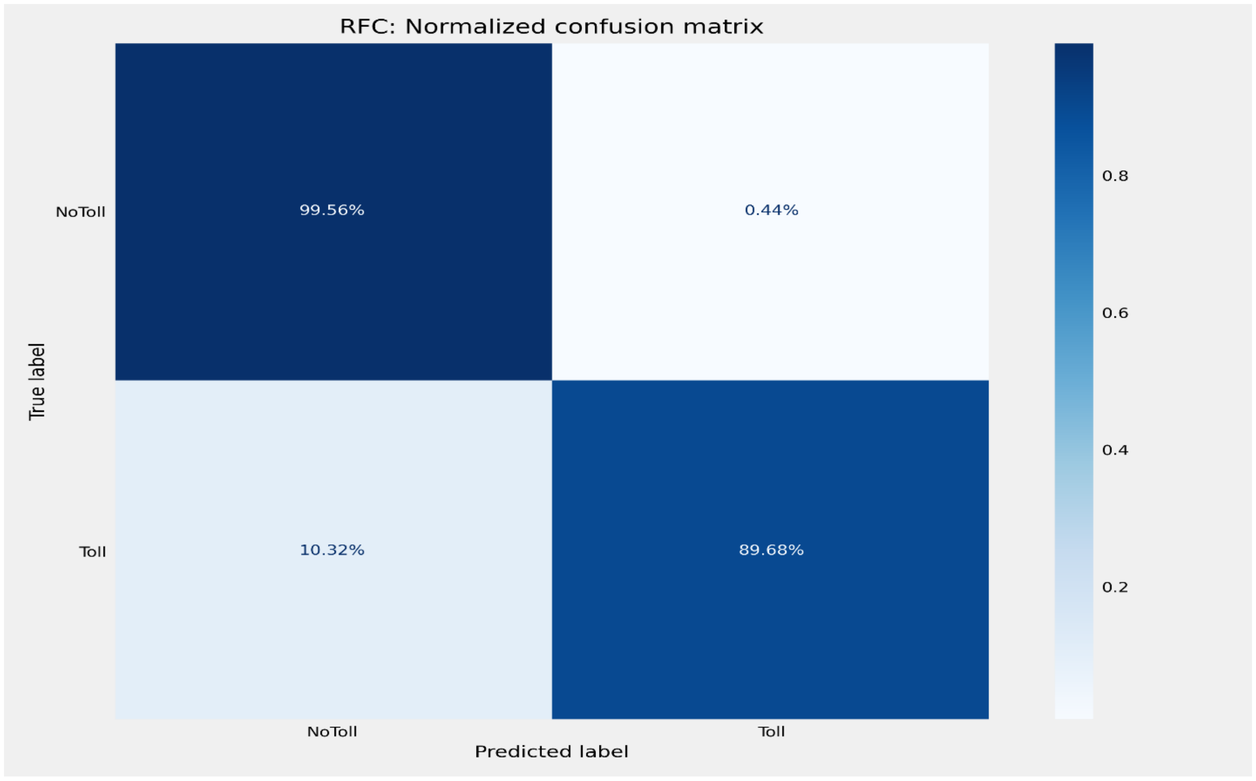


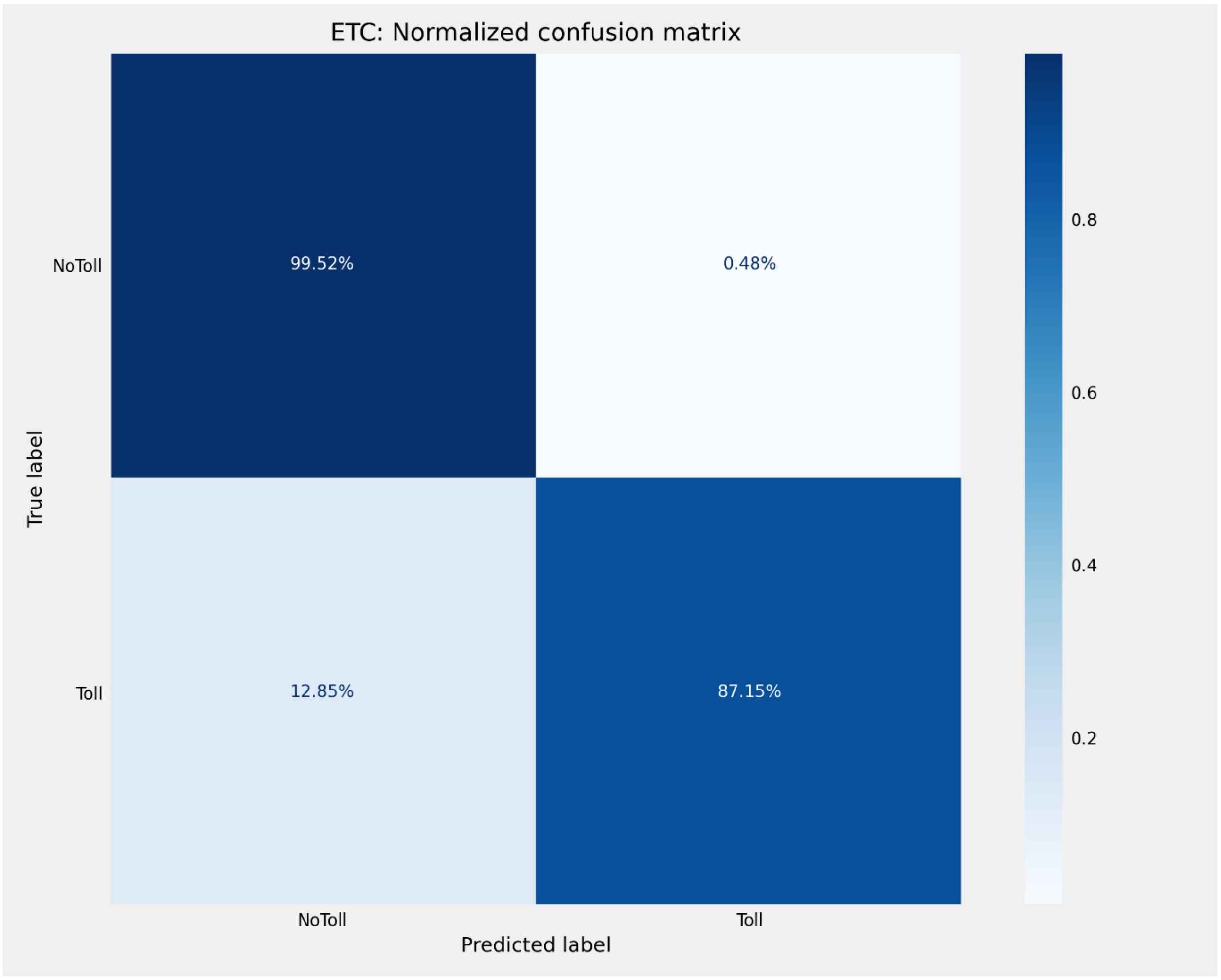
Model Selection

Based on the metrics evaluated and confusion matrix above, we also evaluated the confusion matrix using the test data on the models (highlighted in the model training and evaluation section) that scores above 90% on all metrics:




DTREE: Normalized confusion matrix
True label
NoToll
Toll
99.29%
0.71%
14.12%
85.88%
NoToll
Toll
Predicted label
0.8
0.6
0.4
0.2

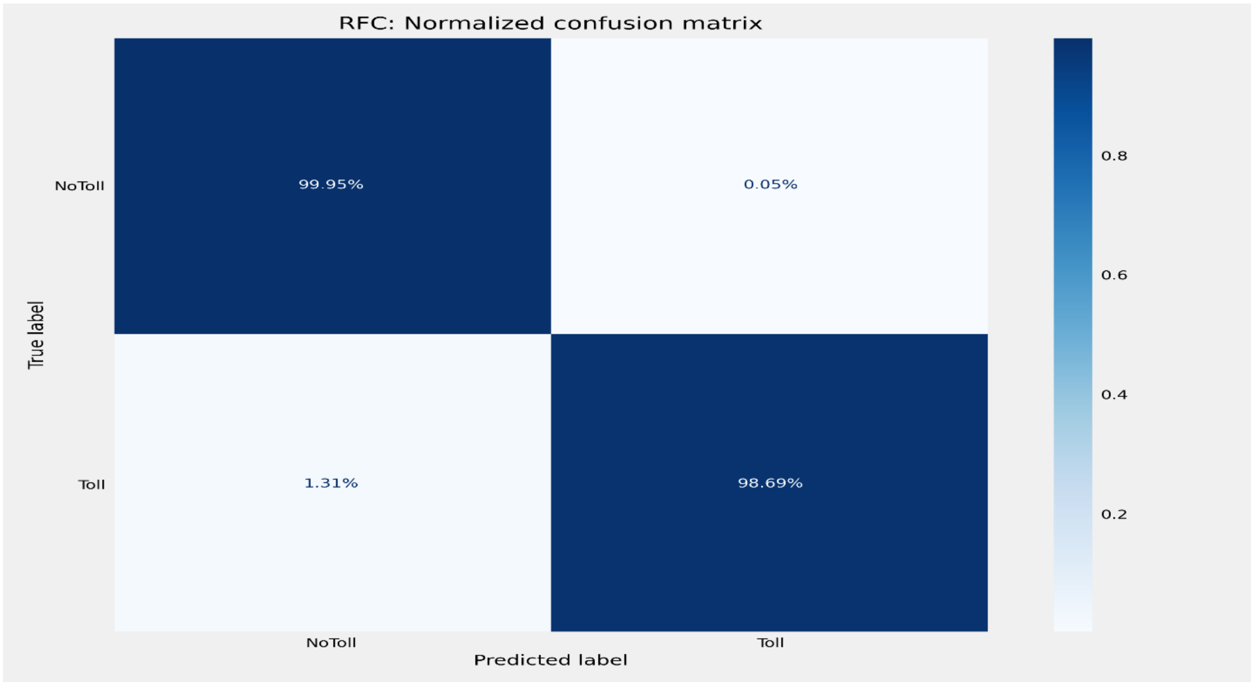

RFC: Normalized confusion matrix
True label
NoToll
Toll
99.56%
0.44%
10.32%
89.68%
NoToll
Toll
Predicted label
0.8
0.6
0.4
0.2

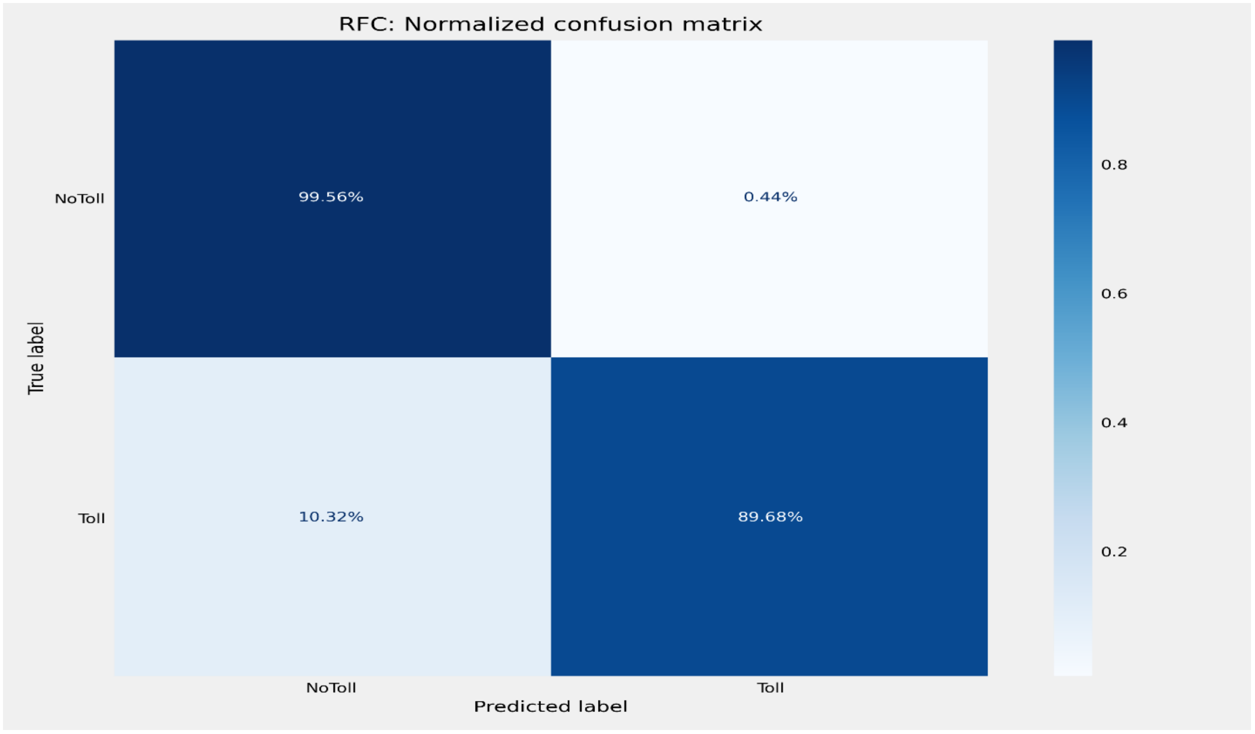


Random Forest Classifier seems to perform better in all metrics and therefore chosen as the best model for our present case.

Next Steps in model improvement

The Random Forest Classifier model chosen above seems to overfit the training data as most of the training data metrics are higher than the test data metric and below confusion matrix shows the same.





Finally, since this data has unbalanced classes, it will be worthwhile to explore techniques such as downsampling, upsampling or a mix of downsampling and upsampling.

Summary

This model will be very useful in predicting Toll or No Toll for every trip and this will help the business to estimate as correct as possible the total amount a customer will be charged. In addition, it will improve transparency as customers will be able to see exactly what they are being charged.